广度搜索

1、广度搜索基础

```
#include<bits/stdc++.h>//1-1751-1      javacn
using namespace std;

int a[110][110];// 表示要存数的地图
int n,m;
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};
int q[10010][3];// 队列，存储每个访问过的点
int k = 1;// 当前填到的数字
int head,tail;// 头尾指针

int main()
{
	cin>>n>>m;
	// 初始化
	head = 1;
	tail = 1;

	// 广搜：先存一个点，作为出发点
	q[1][1] = 1;
	q[1][2] = 1;
	a[1][1] = k;
	k++;
```

```
    //当队列中有点可以访问时
    int tx,ty;
    while(head <= tail)
    {
        //查看队头可以去哪些点
        //看队头的 4 个方向
        for(int i = 1;i <= 4;i++)
        {
            //head 对应点的坐标：q[head][1],q[head][2]
            tx = q[head][1] + fx[i];
            ty = q[head][2] + fy[i];
            //如果这个点可以走：在迷宫内，没走过
            if(tx>=1&&tx<=n&&ty>=1&&ty<=m&&a[tx][ty]==0)
            {
                //走 tx,ty 点：将该点赋值，将该点存入队列
                a[tx][ty] = k;
                k++;
                tail++;                         //将该点存入队列
                q[tail][1] = tx;
                q[tail][2] = ty;
            }
        }
        //当队头对应的 4 个方向的点都讨论完
        //让队头出队，讨论下一个点
        head++;
    }

    for(int i = 1;i <= n;i++){
        for(int j = 1;j <= m;j++){
            cout<<a[i][j]<<" ";
        }
        cout<<endl;
    }
    return 0;
}
```

```cpp
#include<bits/stdc++.h>//1-1751-2   wale
using namespace std;
int a[110][110];//a 存储矩阵
int n,m,i,j;
// 方向数组
int fx[5]={0,0,1,0,-1};
int fy[5]={0,1,0,-1,0};
queue<int> x, y; // 坐标队列
int s=1;
void bfs()
{
	x.push(1);
	y.push(1);
	a[1][1]=s;
	while(!x.empty())
	{
		int tx=x.front(), ty=y.front();
		// 以这一点为基础 找周边的点
		for(i=1; i<=4; i++)
		{
			int nx = tx + fx[i];
			int ny = ty + fy[i];
			if(nx>=1 && nx<=n && ny>=1 && ny<=m && a[nx][ny]==0)
			{
				s++;
				a[nx][ny]=s;
				x.push(nx);
				y.push(ny);
			}
		}
		// 基础点周边的点找完了 基础点出队列
		x.pop();
		y.pop();
	}
}
```

```
int main()
{
	cin>>n>>m;
	bfs(); //广搜

	for(int i = 1; i <= n; i++)
	{
		for( j = 1; j <= m; j++)
		{
			cout<< a[i][j] << " ";
		}
		cout << endl;
	}
}
```

思路：从泉眼出发，广搜求相邻的点中，值 <= 泉眼值的点的数量。

```
#include <iostream>//2-1443-1    javacn
using namespace std;
int n,m,p1,p2;
int a[1010][1010];
bool f[1010][1010];// 标记某个点是否走过
int que[1000100][3];
int fx[5] = {0,0,1,0,-1};// 方向数组
int fy[5] = {0,1,0,-1,0};

int main()
{
	int i,j,head = 1,tail = 1;
	int x,y;
	cin>>n>>m>>p1>>p2;
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= m;j++)
		{
			cin>>a[i][j];
		}
	}

	que[head][1] = p1;
	que[head][2] = p2;
	f[p1][p2] = true;// 标记走过了
```

```
        while(head <= tail)
        {
                for(i = 1;i <= 4;i++)
                {
                        x = que[head][1] + fx[i];
                        y = que[head][2] + fy[i];
                        //判断边界
                        if(x <= n && x >= 1 && y <= m && y >= 1 && a[x][y] <= a[p1]
[p2] && f[x][y] == false)
                        {
                                tail++;
                                que[tail][1] = x;
                                que[tail][2] = y;
                                f[x][y] = true;//标记走过了
                        }
                }

                head++;//统计下一个点的可行路径
        }

        cout<<tail<<endl;
}
```

```cpp
#include<iostream>//2-1443-2   深搜   anselxu
#include<cstring>
#include<algorithm>
#include<cmath>
#include<cstdio>
using namespace std;
int n,m,p1,p2,k,a[1001][1001];
int dx[]={0,1,0,-1};
int dy[]={1,0,-1,0};
int ans=0;
void dfs(int x,int y){
	ans++;
	a[x][y]=k+1;
	int h,l;
	for(int i=0;i<4;i++)
	{
		h=x+dx[i];
		l=y+dy[i];
		if(a[h][l]<=k&&h>0&&l>0&&h<=n&&l<=m)	{	dfs(h,l);	}
	}
}
int main(){
	cin>>n>>m>>p1>>p2;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=m;j++)
		{
			cin>>a[i][j];
		}
	}
	k=a[p1][p2];
	dfs(p1,p2);
	cout<<ans;
	return 0;
}
```

思路：从出发点开始，探测所有可探测的点，看是否有目标点，如果有，就表示可达，否则表示不可达！

走到终点后，我们要将程序直接停掉，来避免不必要的递归：

（1）return：停止当前的函数，如果函数是递归产生的，不会停止所有的递归，只是停止本次函数的递归，退到上一次调用的地方；

（2）exit(0)：停止程序，无论是否有递归，全部停止。

```
#include <bits/stdc++.h>//3-1430-1     javacn
using namespace std;
/*
1. 判断如果起止点有 1，就不能走；
2. 从起点开始搜索，如果走到过终点，标记；
*/
int a[110][110];
int n;
int ha,la,hb,lb;

// 搜索所有可行的点，走过标记
void dfs(int x,int y)
{
	//cout<<x<<" "<<y<<endl;
	a[x][y] = 1;// 走过的点标记
	// 判断是否到达终点
	if(x==hb&&y==lb)
	{
		cout<<"YES";
		exit(0);
	}

	// 判断四个方向，是否有能走的点，如果有，直接递归执行
	if(y+1<=n&&a[x][y+1]==0) dfs(x,y+1);
	if(x+1<=n&&a[x+1][y]==0) dfs(x+1,y);
	if(y-1>=1&&a[x][y-1]==0) dfs(x,y-1);
	if(x-1>=1&&a[x-1][y]==0) dfs(x-1,y);
}
```

```
int main(){
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= n;j++)
		{
			cin>>a[i][j];
		}
	}
	cin>>ha>>la>>hb>>lb;

	// 如果起止点不能走，输出 NO
	if(a[ha][la]==1 || a[hb][lb]==1)
	{
		cout<<"NO";
		return 0;
	}

	dfs(ha,la);// 从 ha,la 开始搜索，走过的点标记为 1
	cout<<"NO";
	return 0;
}
```

```cpp
#include<iostream>//3-1430-2    迷宫出口   jiangyf70
using namespace std;
int a[110][110], r[100100][3];
int fx[] = {0, 1, 0, -1};
int fy[] = {1, 0, -1, 0};
int main()
{
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++) //题目坐标从 1,1 开始的。
            cin >> a[i][j];
    }
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;
    if(a[x1][y1] || a[x2][y2])
    {
        cout << "NO";
        return 0;
    }

    int head = 0, tail = 0;
    r[head][0] = x1;
    r[head][1] = y1;
    a[x1][y1] = 1;
    bool f = false; //是否到终点
```

```
    while(head <= tail && f == false)
    {
        for(int i = 0; i < 4; i++)
        {
            int dx = r[head][0] + fx[i];
            int dy = r[head][1] + fy[i];
            if(dx == x2 && dy == y2)
            {
                f = true;
            }
            else if(dx >= 1 && dx <= n && dy >= 1 && dy <= n && a[dx][dy] == 0)
            {
                tail++;
                r[tail][0] = dx;
                r[tail][1] = dy;
                a[dx][dy] = 1;
                    //cout <<head <<" " << tail <<" " <<  r[tail][0] << " "<<
r[tail][1] << endl;

            }
        }
        head++;
    }
    if(f) cout << "YES";
    else cout <<"NO";
    return 0;
}
```

```cpp
#include<bits/stdc++.h>//4-1434-1    广搜      jiangyf70
using namespace std;
int n, m;
char a[110][110];
int r[100100][3];
int fx[] = {0, 1, 0, -1};
int fy[] = {1, 0, -1, 0};
void bfs(int x, int y)
{
    int head = 0, tail = 0;
    memset(r, 0, sizeof(r));
    r[head][0] = x;
    r[head][1] = y;
    a[x][y] = '.';
    while(head <= tail)
    {
        for(int i = 0; i < 4; i++)
        {
            int dx = r[head][0] + fx[i];
            int dy = r[head][1] + fy[i];
            if(a[dx][dy] == 'W')
            {
                tail++;
                r[tail][0] = dx;
                r[tail][1] = dy;
                a[dx][dy] = '.';
            }
        }
        head++;
    }

}
```

```
void print()
{
    for(int i = 0; i < n; i++)
    {
        cout << a[i] << endl;
    }
    cout << endl;
}

int main()
{
    cin >> n >> m;
    for(int i = 0; i < n; i++)      cin >> a[i];
    int cnt = 0;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            if(a[i][j] =='W')
            {
                cnt++;
                bfs(i, j);
                //print();
            }
        }
    }
    cout << cnt;
    return 0;
}
```

循环每个点，如果当前的点是’W’，则计数器自增 1；然后从当前的 i，j 点开始递归，将上下左右中相邻的为 W 的点全部标记为’.’。

深搜求解：

```
#include <bits/stdc++.h>//4-1434-2    javacn
using namespace std;

/*
  思路：当遇到积水格，计数器加 1，并将该区域的池塘抽干
*/
int n,m;
char a[110][110]; // 默认初值为 '\0'
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};

// 深搜：将 xy 及相邻的积水点全部标记为 .
void dfs(int x,int y)
{
	// 将递归到的有效的点（相邻的有积水的点）标记为 .
	a[x][y] = '.';
	int tx,ty;
	// 递归尝试 4 个方向
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];

		// 如果该点有效（本题判断相邻格是 W，就不用判断是否出边界了）
		if(a[tx][ty] == 'W')
		{
			dfs(tx,ty);
		}
	}
}
```

```
int main()
{
	cin>>n>>m;
	int i,j,c = 0;
	// 读入地图
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= m;j++)
		{
			cin>>a[i][j];
		}
	}

	// 依次遍历每个点，如果是池塘，就将计数器 +1，并将相邻池塘单元格全部标记为点
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= m;j++)
		{
			// 如果该点是池塘
			if(a[i][j] =='W')
			{
				c++;
				dfs(i,j);
			}
		}
	}

	cout<<c;
}
```

```cpp
#include <bits/stdc++.h>//4-1434 -3    不使用方向数组的参考解法：    javacn
using namespace std;

int n,m,c = 0;
char a[110][110];// 存储地图数据

// 将从 x,y 点开始的相邻的 W 标记为 .
void dfs(int x,int y)
{
	a[x][y] = '.';

	// 尝试四个相邻的方向
	// 由于字符数组出了边界不可能是 W，因此字符数组可以不判断出边界
	if(a[x][y+1]=='W') dfs(x,y+1);
	if(a[x+1][y]=='W') dfs(x+1,y);
	if(a[x][y-1]=='W') dfs(x,y-1);
	if(a[x-1][y]=='W') dfs(x-1,y);
}

int main()
{
	// 读入
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cin>>a[i][j];
		}
	}

	// 从每个点开始，遇到 W，C++，并将相邻的能走到的 W 全标记为 .
```

```
    //从每个点开始，遇到 W，C++，并将相邻的能走到的 W 全标记为 .
    for(int i = 1;i <= n;i++)
    {
        for(int j = 1;j <= m;j++)
        {
            if(a[i][j] == 'W')
            {
                c++;
                //从该点开始搜索，将相邻的 W 标记为 .
                dfs(i,j);
            }
        }
    }

    cout<<c<<endl;
  return 0;
}
```

```cpp
#include<bits/stdc++.h>//5-1907-1    广搜    jiangyf70
using namespace std;
int n, m;
char a[110][110];
int  r[100100][3];
int fx[] = {0, 1, 0, -1};
int fy[] = {1, 0, -1, 0};

void bfs(int x, int y)
{
    memset(r, 0, sizeof(r));
    int head = 0, tail = 0;
    r[head][0] = x;
    r[head][1] = y;
    a[x][y] = '0';
    while(head <= tail)
    {
        for(int i = 0; i < 4; i++)
        {
            int dx = r[head][0] + fx[i];
            int dy = r[head][1] + fy[i];
            if(a[dx][dy] > '0' && a[dx][dy] <='9')
            {
                tail++;
                r[tail][0] = dx;
                r[tail][1] = dy;
                a[dx][dy] = '0';
            }
        }
        head++;
    }

}
```

```
void print()
{
    for(int i = 0; i < n; i++) cout << a[i] << endl;
    cout << endl;
}

int main()
{
    cin >> n >> m;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            cin >> a[i][j];
        }
    }
    int cnt = 0;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            if(a[i][j] > '0' && a[i][j] <= '9')
            {
                cnt++;
                bfs(i, j);
                //print();
            }
        }
    }
    cout << cnt;
    return 0;
}
```

```cpp
#include <bits/stdc++.h>//5-1907-2    深搜    javacn
using namespace std;

int n,m,c;
char a[110][110];
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};

void dfs(int x,int y)
{
    a[x][y] = '0';

    int tx,ty;
    for(int i = 1;i <= 4;i++)
    {
        tx = x + fx[i];
        ty = y + fy[i];

        if(a[tx][ty]>='1' && a[tx][ty]<='9')
        {
            dfs(tx,ty);
        }
    }
}
```

```
int main()
{
    cin>>n>>m;

    for(int i = 1;i <= n;i++)
    {
        for(int j = 1;j <= m;j++)
        {
            cin>>a[i][j];
        }
    }

    for(int i = 1;i <= n;i++)
    {
        for(int j = 1;j <= m;j++)
        {
            if(a[i][j]>='1' && a[i][j]<='9')
            {
                c++;
                dfs(i,j);
            }
        }
    }

    cout<<c;
    return 0;
}
```

```cpp
#include<iostream>//6-1897-1  jiangyf70
using namespace std;

int n, m;
char a[30][30];
int r[1000][3];
int fx[] = {0, 1, 0, -1};
int fy[] = {1, 0, -1, 0};

int main()
{
    cin >> m >> n;
    int x, y;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            cin >> a[i][j];
            if(a[i][j] == '@')
            {
                x = i;
                y = j;
            }
        }
    }

    int head = 1, tail = 1;
    r[head][0] = x;
    r[head][1] = y;
    a[x][y] = '#';
```

```
    while(head <= tail)
    {
        for(int i = 0; i < 4; i++)
        {
            int dx = r[head][0] + fx[i];
            int dy = r[head][1] + fy[i];
            if(a[dx][dy] =='.')
            {
                tail++;
                r[tail][0] = dx;
                r[tail][1] = dy;
                a[dx][dy] = '#';

            }
        }
        head++;
    }
    cout << tail;

    return 0;
}
```

1、读入数据，读入的同时记录 @ 的位置（下标）

2、从 @ 的位置出发，深搜，每搜过一个点，通过公共计数器 c 来统计搜索的次数

注意：为了防止死循环，走过一个点，就将该点从 . 改成 #

```
#include<bits/stdc++.h>//6-1897-2
using namespace std;

int n,m,s1,s2;//s1,s2 代表出发点的位置
char a[50][50];
int c = 0;

// 深搜从出发点开始能够访问到的点，并标记为 #
void dfs(int x,int y)
{
	a[x][y] = '#';// 标记走过，防止死循环
	c++;// 走过一个点计数一次

	// 尝试四方向
	// 不需要判断出边界，因为出了边界不可能是 .
	if(a[x][y+1]=='.') dfs(x,y+1);
	if(a[x+1][y]=='.') dfs(x+1,y);
	if(a[x][y-1]=='.') dfs(x,y-1);
	if(a[x-1][y]=='.') dfs(x-1,y);
}
```

```
int main()
{
	// 先读入列，再读入行
	cin>>m>>n;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cin>>a[i][j];
			// 记录出发点的坐标
			if(a[i][j] == '@')
			{
				s1 = i;
				s2 = j;
			}
		}
	}

	// 从出发点开始搜索
	dfs(s1,s2);
	cout<<c;
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//7-1966-1  人造星空 jiangyf70
using namespace std;
int n, m;
char a[110][110];
int r[110110][3];
int fx[] ={0, 0, 0, 0, 1, 2, -1, -2, 1, 1, -1, -1};
int fy[] ={1, 2, -1, -2, 0, 0, 0, 0, 1, -1, 1, -1};

void bfs(int x, int y)
{
    memset(r, 0, sizeof(r));
    int head = 0, tail = 0;
    r[head][0] = x;
    r[head][1] = y;
    a[x][y] = '-';
    while(head <= tail)
    {
        for(int i = 0; i < 12; i++)
        {
            int dx = r[head][0] + fx[i];
            int dy = r[head][1] + fy[i];
            if(a[dx][dy] == '#')
            {
                tail++;
                r[tail][0] = dx;
                r[tail][1] = dy;
                a[dx][dy] = '-';
            }
        }
        head++;
    }
}
```

```
int main()
{
    cin >> n >> m;
    for(int i = 0; i < n; i++) cin >> a[i];
    int cnt = 0;
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            if(a[i][j] == '#')
            {
                cnt++;
                bfs(i, j);
            }
        }
    }
    cout << cnt;
    return 0;
}
```

本题求解：满足曼哈顿距离 <=2 的情况下的连通块的数量

距离 x,y 点，满足曼哈顿距离 <=2 的点，有 12 个

```
#include<bits/stdc++.h>//7-1966-2    javacn
using namespace std;
int n,m;
char a[110][110];
int c = 0;// 统计有多少个连通块

// 存储相对 xy 点曼哈顿距离 <=2 的相对位置的差值
int fx[13] = {0,-2,-1,-1,-1,0,0,0,0,1,1,1,2};
int fy[13] = {0,0,-1,0,1,-2,-1,1,2,-1,0,1,0};

// 将 xy 相邻 12 个方向的能走到的 #，深搜标记为 -
void dfs(int x,int y)
{
	a[x][y] = '-';
	int tx,ty;
	for(int i = 1;i <= 12;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];
		// 出边界不可能是 #
		if(a[tx][ty] == '#')
		{
			dfs(tx,ty);
		}
	}
}
```

```
int main()
{
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cin>>a[i][j];
		}
	}

	// 遍历每个点，如果是 #，深搜将 12 方向相邻的 # 都标记为 -
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			if(a[i][j] == '#')
			{
				c++;
				dfs(i,j);// 从 ij 开始搜索标记
			}
		}
	}

	cout<<c;
	return 0;
}
```

1、广度最少步数最短路径

深搜求最少步数：

思路：

1、准备一个整数数组，记录从出发点到每个点至少需要多少步，初始化为 INT_MAX；

2、从出发点开始探测，顺时针探测，如果该点可达，且到该点的步数更少，则替换 d 数组的步数；

3、最终 d 数组记录了到每个点至少需要多少步，d[n][m] 就是最终结果；

```
#include <bits/stdc++.h>//1-1432-1    javacn
using namespace std;
int n,m;
char a[50][50];// 地图
int d[50][50];// 存储走到每个点最少需要多少步
// 方向值变化的数组
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};
```

```
// 递归探索地图，求到走到每个点最少需要多少步
void dfs(int x,int y,int dep)
{
	d[x][y] = dep;
	int tx,ty;
	// 循环数组，得到 4 个新的方向，递归探索 4 个方向
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];
		// 如果 tx,ty 可以探索（该点在地图内，且该点是 .，且走到该点的步数更少）
		if(a[tx][ty]=='.'&&dep+1<d[tx][ty])
		{
			dfs(tx,ty,dep+1);
		}
	}
}
int main()
{
	int i,j;
	cin>>n>>m;
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= m;j++)
		{
			cin>>a[i][j];
			// 将最少步数初始值设为 INT_MAX
			d[i][j] = INT_MAX;
		}
	}
	// 调用函数
	dfs(1,1,1);
	cout<<d[n][m];
}
```

题目数据范围要求 r,c<=40，实际时超过的，开始我提交开的数组 a 为 45，报运行错误。开到 55 提交就 A 了。

```
#include<iostream>//1-1432-2    anselxu
using namespace std;
struct bfs_arr{
	int x,y,t,pr;
};
bfs_arr b[5000];
char a[55][55];
int main(){
	int dx[]={0,1,0,-1};
	int dy[]={1,0,-1,0};
	bool f=0;
	int n,m,h,l;
	cin>>n>>m;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=m;j++)
		{
			cin>>a[i][j];
		}
	}
	int head,tail;
	head=1;
	tail=1;
	b[head].x=1;
	b[head].y=1;
	b[head].t=1;
```

```
	while(head<=tail)
	{
		h=b[head].x;
		l=b[head].y;
		a[h][l]='#';
		for(int i=0;i<4;i++)
		{
			if(a[h+dx[i]][l+dy[i]]=='.')
			{
				a[h+dx[i]][l+dy[i]]='#';
				b[++tail].t=b[head].t+1;
				b[tail].x=h+dx[i];
				b[tail].y=l+dy[i];
				b[tail].pr=head;
				if(h+dx[i]==n&&l+dy[i]==m)
				{
					cout<<b[tail].t;
					f=1;
					break;
				}
			}
		}
		head++;
		if(f) break;
	}
	return 0;
}
```

```cpp
#include <iostream>//2-1433-1  走出迷宫最少步数   kevinh
#include <climits>
using namespace std;

int n,m,r[110][110];
char a[110][110];
int bx,by,ex,ey;
int fx[4][2] = {{-1,0},{0,1},{1,0},{0,-1}};

void dfs(int x,int y,int dep)
{
	r[x][y]=dep;
	int tx,ty;
	for(int i=0;i<4;i++)
	{
		tx = x+fx[i][0];
		ty = y+fx[i][1];

		if(a[tx][ty]=='.'&&dep+1<r[tx][ty])
		{
			dfs(tx,ty,dep+1);
		}
	}
}
```

```
int main()
{
	int i,j;
	cin>>n>>m;
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=m;j++)
		{
			cin>>a[i][j];
			r[i][j] = INT&#95;MAX;
				// 找出起点和出口，并设置为.
			if(a[i][j]=='S')
			{
				bx=i;
				by=j;
				a[i][j]='.';
			}
			else if(a[i][j]=='T')
			{
				ex=i;
				ey=j;
				a[i][j]='.';
			}

		}
	}
	dfs(bx,by,0);
	cout<<r[ex][ey];
	return 0;
}
```

```
include<bits/stdc++.h>//2-1433-2    Mihui0705
using namespace std;
int n,m,s1,s2,e1,e2;
char a[50][50];
int d[50][50];
int fx[4]={0,1,0,-1};
int fy[4]={1,0,-1,0};
void dfs(int x,int y,int k)
{

      d[x][y]=k;
      int tx,ty;
      for(int i=0;i<4;i++)
      {
            tx=x+fx[i];
            ty=y+fy[i];
            if((a[tx][ty]=='.'||a[tx][ty]=='T' )&& k+1<d[tx][ty])//   条件 1：是空地 条件 2：新获得的路径值比之前存储的最小值小（大的就不用更改了）
            {
                  dfs(tx,ty,k+1);//     每次往后面走都是前面一格最小值 +1（因为是通路）
            }
      }
}
```

```
int main()
{

	cin>>n>>m;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=m;j++)
		{
			cin>>a[i][j];
			if(a[i][j]=='S')
			{
				s1=i;
				s2=j;
			}
			if(a[i][j]=='T')
			{
				e1=i;
				e2=j;
			}
		}
	}
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=m;j++)
		{
			d[i][j]=INT&#95;MAX;// 要更改存储最小值，有一个比较存在，就得让其初始值处于无限大的状态
		}
	}
	dfs(s1,s2,0);
	cout<<d[e1][e2];
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//3-1900    采药的最短路径
using namespace std;
char a[30][30];
int b[30][30];
int n, m;
int dx[4] = {0, 1, 0, -1};
int dy[4] = {1, 0, -1, 0};

void dfs (int x, int y, int dep)
{
    b[x][y] = dep;
    for(int i = 0; i < 4; i++)
    {
        int fx = x + dx[i];
        int fy = y + dy[i];
        if((a[fx][fy] == '.' || a[fx][fy] == '*') && b[x][y] + 1 < b[fx][fy])
        {
            dfs(fx, fy, dep+1);
        }
    }
}
```

```
int main()
{
    int x1, x2, y1, y2;
    cin >> n >> m;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= m; j++)
        {
            cin >> a[i][j];
            b[i][j] = INT&#95;MAX;
            if(a[i][j] == '@')
            {
                x1 = i;
                y1 = j;
            }
            if(a[i][j] == '*')
            {
                x2 = i;
                y2 = j;
            }
        }

    }
    dfs(x1, y1, 1);
    if(b[x2][y2] < INT&#95;MAX) cout << b[x2][y2] - 1; //不算最后一步。
    else cout << -1;


    return 0;
}
```

```cpp
#include <bits/stdc++.h>//4-2109 古希腊之争 javacn
using namespace std;

int q[250010][4];// 队列
int n,m,c,h,t;
char a[510][510];
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};
int s1,s2,e1,e2;

int main()
{
	cin>>m>>n>>c;
	for(int i = 1; i <= n; i++)
	{
		for(int j = 1; j <= m; j++)
		{
			cin>>a[i][j];
			if(a[i][j] == 'S')
			{
				s1 = i;
				s2 = j;
				a[i][j] = '.';
			}
			else if(a[i][j] == 'E')
			{
				e1 = i;
				e2 = j;
				a[i][j] = '.';
			}

		}
	}
```

```
        //bfs：初始化
        h = 1;
        t = 1;
        q[1][1] = s1;
        q[1][2] = s2;
        q[1][3] = 0;
        a[s1][s2] =  '#' ;// 走过标记
        int tx,ty;
        while(h <= t)
        {
              for(int i = 1; i <= 4; i++)
              {
                    tx = q[h][1] + fx[i];
                    ty = q[h][2] + fy[i];
                    // 该点可行
                    if(a[tx][ty]=='.')
                    {
                          // 入队
                          t++;
                          q[t][1] = tx;
                          q[t][2] = ty;
                          q[t][3] = q[h][3] + 1;
                          a[tx][ty] = '#';// 标记
                          if(tx==e1&&ty==e2)       // 判终点
                          {
                                cout<<q[t][3] * c;
                                return 0;
                          }
                    }
              }
              h++;
        }
        cout<<-1;
        return 0;
}
```

最少步数问题：

```
#include<bits/stdc++.h>//5-1438    骑士巡游      javacn
using namespace std;
int d[11][11];// 存储最少步数
int fx[9]= {0,-1,-2,1,-2,2,2,-1,1};
int fy[9]= {0,-2,-1,-2,1,-1,1,2,2};
int n,m;
//dfs 求走到每个点的最少步数
void dfs(int x,int y,int step)
{

    d[x][y]=step;// 更新最少步数
    for(int i=1; i<=8; i++)
    {
        int tx = x+fx[i];
        int ty = y+fy[i];
        if(tx>=1&&tx<=n&&ty>=1&&ty<=m&&step+1<d[tx][ty])
        {
            dfs(tx,ty,step+1);
        }

    }
}
```

```
int main()
{
    int x,y,s,t;
    cin>>n>>m>>x>>y>>s>>t;
    // 初始化 d 数组为无穷大
    for(int i=1; i<=n; i++)
    {
        for(int j=1; j<=m; j++)
        {
            d[i][j]=INT_MAX;
        }
    }
    dfs(x,y,0);
    cout<<d[s][t];
    return 0;
}
```

```
#include <bits/stdc++.h>//6-1442  走出迷宫最短路径  javacn
using namespace std;

/*
  思路：从出发点开始广搜，记录每个点及每个点的前驱点（父节点）
*/
int n,m,i,j;
int a[150][150];// 迷宫
int q[40000][4];// 队列
int fx[5] = {0,0,1,0,-1};// 方向数组
int fy[5] = {0,1,0,-1,0};
int tx,ty;// 表示从 head 开始探索的新方向
int head = 1,tail = 1;
int s1,s2,e1,e2;// 起止点

// 打印路径
void print(int k)
{
	// 如果前驱节点不为 0（有前驱节点，则递归找前驱节点）
	if(q[k][3] != 0)
	{
		print(q[k][3]);
	}

	cout<<"("<<q[k][1]<<","<<q[k][2]<<")";
	if(k != tail)
	{
		cout<<"->";
	}
}
```

```
int main()
{
	cin>>n>>m;
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= m;j++)
		{
			cin>>a[i][j];
		}
	}

	// 读入起止点
	cin>>s1>>s2>>e1>>e2;

	// 出发点直接存入队列
	q[1][1] = s1;
	q[1][2] = s2;
	// 出发点没有前驱点（父节点）
	q[1][3] = 0;
```

```
    while(head <= tail)
    {
        // 遍历 head 的四个方向
        for(i = 1;i <= 4;i++)
        {
            // 得到 head 节点的四个方向的坐标
            tx = q[head][1] + fx[i];
            ty = q[head][2] + fy[i];

            // 如果该方向可达（在迷宫内，且没走过）
            if(tx>=1&&tx<=n&&ty>=1&&ty<=m&&a[tx][ty]==0)
            {
                // 走过的点做标记
                a[tx][ty] = 1;
                // 将点存入队列
                tail++;
                q[tail][1] = tx;
                q[tail][2] = ty;
                // 记录该点的前驱点的行下标
                q[tail][3] = head;

                // 如果到了终点，打印
                if(tx == e1 && ty == e2)
                {
                    print(tail);
                    return 0;
                }
            }
        }
        head++;
    }

    cout<<"no way";
    return 0;
}
```

注意 bfs 要特判起点 == 终点的情况，而 dfs 是不需要特判这种情况的。

```
#include<bits/stdc++.h>//7-1819-1    奇怪的电梯    javacn
using namespace std;

int n;
int q[210][3];// 走到每个点及最少步数
int h,t;
int fx[3] = {0,1,-1};
int a[210];
bool f[210];// 标记走过
int s,e;// 起止点

int main()
{
	cin>>n>>s>>e;
	for(int i = 1;i <= n;i++)
	{
		cin>>a[i];
	}

	// 特判起点 == 终点的情况
	if(s == e)
	{
		cout<<0;
		return 0;
	}

	//bfs 求从 s 点到 e 点的最少步数
	h = 1;
	t = 1;
	q[1][1] = s;
	q[1][2] = 0;
	f[s] = true;// 走过标记
```

```
	int tx,ty;// 表示要去的点
	while(h <= t)
	{
		for(int i = 1;i <= 2;i++)
		{
			tx = q[h][1] + fx[i] * a[q[h][1]];
			// 如果该层可以走
			if(tx>=1&&tx<=n&&f[tx]==false)
			{
				// 入队，标记，判终点
				t++;
				q[t][1] = tx;
				q[t][2] = q[h][2] + 1;
				f[tx] = true;

				// 判终点
				if(tx == e)
				{
					cout<<q[t][2];
					return 0;
				}
			}
		}

		h++;
	}

	cout<<-1;
	return 0;
}
```

本题本质上来说还是最少步数问题，按几次按钮，相当于最少要走多少步，只是本题不同于迷宫类问题，本题只有 2 个方向，上或者下。

```
#include<bits/stdc++.h>/7-1819-2     javacn
using namespace std;

int a[210];
int d[210];// 表示走到每一层最少要按几次按钮
int n;
int s,e;// 代表出发楼层和要到达的楼层

int fx[3] = {0,1,-1};

// 走到 x 楼，最少需要 step 步
void dfs(int x,int step)
{
	d[x] = step;
	int to;
	// 尝试两个方向
	for(int i = 1;i <= 2;i++)
	{
		to = x + fx[i] * a[x];
		// 如果该楼层可以去
		if(to>=1&&to<=n&&step+1<d[to])
		{
			dfs(to,step+1);
		}
	}
}
```

```
int main()
{
	cin>>n>>s>>e;
	for(int i = 1;i <= n;i++)
	{
		cin>>a[i];
		d[i] = INT_MAX;
	}

	// 从 s 点开始搜索
	dfs(s,0);

	if(d[e] == INT_MAX) cout<<-1;
	else cout<<d[e];
	return 0;
}
```

```cpp
#include<iostream>//8-1441-1    anselxu
using namespace std;
struct pp{int a,b,t,pr;};
int dx[]={2,2,1,1,-2,-2,-1,-1};
int dy[]={1,-1,2,-2,1,-1,2,-2};
char b[180][180];
pp a[40001];
int main()
{
	int n,m,x,y;
	cin>>m>>n;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=m;j++)
		{
			cin>>b[i][j];
			if(b[i][j]=='K')
			{
				x=i;
				y=j;
			}
		}
	}
	int head,tail;
	head=1;
	tail=1;
	a[head].a=x;
	a[head].b=y;
	a[head].t=0;
	b[x][y]='*';
```

```
	while(head<=tail)
	{
		int h,l;
		x=a[head].a;
		y=a[head].b;
		for(int i=0;i<8;i++)
		{
			h=x+dx[i];
			l=y+dy[i];
			if(h<=n&&h>0&&l<=m&&l>0&&b[h][l]!='*')
			{
				tail++;
				a[tail].a=h;
				a[tail].b=l;
				a[tail].t=a[head].t+1;
				a[tail].pr=head;
				if(b[h][l]=='H')
				{
					cout<<a[tail].t;
					return 0;
				}
				b[h][l]='*';
			}
		}
		head++;
	}
	return 0;
}
```

```cpp
#include <bits/stdc++.h>//8-1441-2  jiangyf70
using namespace std;    // 注意 n, m 的输入是颠倒的
char a[155][155];
int b[155][155];
int fx[8] = {-2, -2, -1, -1, 1, 1, 2, 2};
int fy[8] = {1, -1, 2, -2, 2, -2, 1, -1};
int n, m;
void dfs(int x, int y, int k)
{
    b[x][y] = k;
    for(int i = 0; i < 8; i++)
    {
        int dx = x + fx[i];
        int dy = y + fy[i];
         if((a[dx][dy] =='.' || a[dx][dy] == 'H') && k + 1 < b[dx][dy]) dfs(dx,
dy, k+1);
    }
}
int main() {
    int x1, y1, x2, y2;
    cin >> n >> m;
    for(int i = 1; i <= m; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            cin >> a[i][j];
            b[i][j] = INT_MAX;
            if(a[i][j] ==  'H' )x2 = i, y2 = j;
            if(a[i][j] ==  'K' )x1 = i, y1 = j;
        }
    }
    dfs(x1, y1, 0);
    cout << b[x2][y2];
    return 0;
}
```

深搜求走到每个点的最少步数，注意本题要先读列，再读行：

```
#include <bits/stdc++.h>//8-1441-3  骑士牛  javacn
using namespace std;

//s1,s2:出发点，e1,e2:终点
int n,m,s1,s2,e1,e2;
char a[200][200];//迷宫
int d[200][200];//走到每个点的最少步数
//方向数组
int fx[10]={0,-2,-2,-1,1,2,2,1,-1};
int fy[10]={0,-1,1,2,2,1,-1,-2,-2};

//深搜求走到每个点的最少步数
void dfs(int x,int y,int step)
{
	d[x][y]=step;
	int tx,ty;
	for(int i=1;i<=8;i++)
	{
		tx=x+fx[i];
		ty=y+fy[i];
		if((a[tx][ty]=='.'||a[tx][ty]=='H')&&step+1<d[tx][ty])
		{
			dfs(tx,ty,step+1);
		}
	}
}
```

```
int main()
{
	// 本题先读列，再读行
	cin>>m>>n;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=m;j++)
		{
			cin>>a[i][j];
			d[i][j]=INT_MAX;
			if(a[i][j]=='K')
			{
				s1=i;
				s2=j;
			}
			else if(a[i][j]=='H')
			{
				e1=i;
				e2=j;
			}
		}
	}
    dfs(s1,s2,0);
    cout<<d[e1][e2];
}
```

3、广度搜索综合

本题由于数据量比较大，因此不能每次查询都从每个点出发，查询能走到的点的数量。可以利用连通块的特性：连通块中从任何一个点出发能走到的点的数量都是连通块的面积。因此可以用一个数组来存储从某个点出发能走到的点的数量，一旦求出某个连通块中点的数量，就将这个连通块中从每个点出发能走到的点的数量都赋值。

这样就避免了重复的查询，参考解法如下：

```
#include <bits/stdc++.h>//1-1803  01 迷宫     javacn
using namespace std;

/*
求解原理：从连通块中的每个点出发，最多能走到的点的数量都是一样的
*/
char a[1010][1010];
int q[1000010][3],h,t;// 存储走过的点
int d[1010][1010];// 存储从每个点出发最多能走到几个点
bool f[1010][1010];
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};
int n,m;//n*n 的迷宫，m 次询问
```

```
// 求：从某个点出发能走到几个点，同时将连通块中从每个点出发能走到几个点，记录好
void bfs(int x,int y)
{
    // 初始化
      h = 1,t = 1;
      q[1][1] = x;
      q[1][2] = y;
      f[x][y] = true;// 走过标记
      int tx,ty;
      while(h <= t)
      {
            for(int i = 1;i <= 4;i++)
            {
                  tx = q[h][1] + fx[i];
                  ty = q[h][2] + fy[i];
                  // 不能出迷宫，没有走过，且值不一样
                  if(tx>=1&&tx<=n&&ty>=1&&ty<=n&&f[tx][ty]==false&&a[tx]
[ty]!=a[q[h][1]][q[h][2]])
                  {
                              // 入队，标记
                        t++;
                        q[t][1] = tx;
                        q[t][2] = ty;

                        f[tx][ty] = true;
                  }
            }
            h++;
      }
    // 循环队列，设置从队列中每个点出发能走到的点的数量都是队列长度
      for(int i = 1;i <= t;i++)
      {
            d[q[i][1]][q[i][2]] = t;
      }
}
```

```
int main()
{
      cin>>n>>m;
    // 读入地图
      for(int i = 1;i <= n;i++)
      {
            scanf("%s",a[i]+1);
      }

    // 读入 m 次询问
      int x,y;
      for(int i = 1;i <= m;i++)
      {
            scanf("%d%d",&x,&y);
        // 判断从 xy 出发能走到几个点，是否求过
            if(d[x][y] == 0) bfs(x,y);

            printf("%d\n",d[x][y]);
      }
      return 0;
}
```

反过来考虑这个问题，判断在圈里面不容易，但判断在圈外面很容易

沿着最外层遍历一圈，从最外层的 0 开始搜索，可以访问到所有封闭的圈以外的 0，标记为 3；

打印：3 打印为 0，1 还是打印为 1，0 打印为 2；

```
#include <bits/stdc++.h>//2-1802-1  填涂颜色 javacn
using namespace std;
int n;
int a[40][40];
// 方向值的变化数组
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};

// 深搜：从 xy 点开始搜索，将所有相邻的 0 标记为 3
void dfs(int x,int y)
{
	a[x][y] = 3;
	// 遍历四方向
	int tx,ty;
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];
		// 判断在方阵内，且要去的点是 0
		if(tx>=1&&tx<=n&&ty>=1&&ty<=n&&a[tx][ty]==0) dfs(tx,ty);
	}
}
```

```
int main()
{
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= n;j++)
		{
			cin>>a[i][j];
		}
	}

	// 遍历最外层的值，找到 0，深搜
	// 第 1 行和最后一行，一定在最外层，其余每行只有第 1 个和最后一个在最外层
	// 循环每一行
	for(int i = 1;i <= n;i++)
	{
		// 只有第 1 行和最后一行要循环每一列
		if(i == 1 || i == n)
		{
			for(int j = 1;j <= n;j++)
			{
				if(a[i][j] == 0) dfs(i,j);
			}
		}
		else
		{
			// 看第 1 个和最后一个值
			if(a[i][1] == 0) dfs(i,1);
			if(a[i][n] == 0) dfs(i,n);
		}
	}

	// 打印
```

```
	//打印
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= n;j++)
		{
			if(a[i][j] == 3) cout<<0<<" ";
			else if(a[i][j] == 0) cout<<2<<" ";
			else cout<<1<<" ";
		}
		cout<<endl;
	}

	return 0;
}
```

```cpp
#include <bits/stdc++.h>//2-1802 -2   R_S
using namespace std;
int a[35][35];
int n;
void dfs(int x ,int y)
{
	a[x][y] = 2;
	int xx[5] = {0,0,1,0,-1};
	int yy[5] = {0,1,0,-1,0};
	for(int i=1;i<=4;i++)
	{
		int tx = x + xx[i];
		int ty = y + yy[i];
		if(tx>=1 && tx<=n-1 && ty>=1 && ty<=n-1 && a[tx][ty]==0)
		{
			dfs(tx,ty);
		}
	}
}
void p()
{
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=n;j++)
		{
			cout <<  a[i][j] << ' ';
		}
		cout << endl;
	}
}
```

```
int main()
{
	cin >> n;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=n;j++)
		{
			cin >> a[i][j];
		}
	}

	// 开始找第一个为 1 的点
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=n;j++)
		{
			if(a[i][j]==1)
			{
				dfs(i+1,j+1);
				p();
				return 0;
			}
		}
	}

	return 0;
}
```

```
/*
1、从第一个点开始找，找到第一个为 1 的点开始深搜，
   因为数字 1 构成圆环，那么最后一个遍历点会回到第一个 1 的位置。
   构成位置后，从倒数第一个点的右边开始填色，填色同样采用深搜的方式。
*/
#include<bits/stdc++.h>//2-1802-3    jiangyf70
using namespace std;
int n;
int a[40][40];
int fx[] = {0, 1, 0, -1};
int fy[] = {1, 0, -1, 0};

void dfs(int x, int y, int k)
{
    a[x][y] = k;
    for(int i = 0; i < 4; i++)
    {
        int dx = x + fx[i];
        int dy = y + fy[i];
        if(a[dx][dy] == 0 && dx >= 1 && dx <= n && dy >= 1 && dy <= n)
            dfs(dx, dy, k);
    }
}
```

```cpp
int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            cin >> a[i][j];
        }
    }

    for(int i = 1; i <= n; i++)
    {
        if(a[1][i] == 0) dfs(1, i, 3);
        if(a[i][1] == 0) dfs(i, 1, 3);
        if(a[n][i] == 0) dfs(n, i, 3);
        if(a[i][n] == 0) dfs(i, n, 3);
    }

    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            if(a[i][j] == 3) a[i][j] = 0;
            else if(a[i][j] == 0) a[i][j] = 2;
            cout << a[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

```cpp
#include<iostream>//2-1802-4  jiangyf70
using namespace std;
int n, a[30][30];
int fx[4] = {0, 1, 0, -1};
int fy[4] = {1, 0, -1, 0};
void dfs(int x, int y)
{
    a[x][y] = 3;
    for(int i = 0; i < 4; i++)
    {
        int dx = x + fx[i];
        int dy = y + fy[i];
        if(dx <= n && dx >= 1 && dy <= n && dy >= 1 && a[dx][dy] == 0)
            dfs(dx, dy);
    }
}
```

```cpp
int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            cin >> a[i][j];

    for(int i = 1; i <= n; i++)
    {
        if(i == 1 || i == n)
        {
            for(int j = 1; j <= n; j++)
            {
                if(a[i][j] == 0)dfs(i, j);
            }
        }
        else
        {
            if(a[i][1] == 0) dfs(i, 1);
            if(a[i][n] == 0) dfs(i, n);
        }
    }

    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            if(a[i][j] == 0) a[i][j] = 2;
            if(a[i][j] == 3) a[i][j] = 0;
            cout << a[i][j] <<" ";
        }
        cout << endl;
    }
    return 0;
}
```

广搜，使用 while 循环从队列头部对应的点一直向一个方向走，如果该点能走，判断该点不在队列中，则入队，第一次走到终点时，对应的拐弯次数，就是最小拐弯次数。

```
#include <bits/stdc++.h>//3-1444    最小拐弯路径    javacn
using namespace std;

const int N = 1e3 + 10;
int a[N][N];
bool f[N][N];
int q[N*N+10][4];
int s1,s2,e1,e2,n,m;
int h,t;
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};

int main()
{
	cin>>n>>m;
	memset(a,-1,sizeof(a));// 设定边界
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cin>>a[i][j];
		}
	}
	cin>>s1>>s2>>e1>>e2;
	h = 1;
	t = 1;
	q[1][1] = s1;
	q[1][2] = s2;
	f[s1][s2] = true;
	q[1][3] = 0;// 拐弯次数
```

```
	while(h <= t)
	{
		for(int i = 1;i <= 4;i++)
		{
			int x = q[h][1];
			int y = q[h][2];// 头部对应的出发点
			while(a[x+fx[i]][y+fy[i]]==0)
			{
				// 如果该点没走过，存储队列
				if(f[x+fx[i]][y+fy[i]]==false)
				{
					t++;
					q[t][1] = x + fx[i];
					q[t][2] = y + fy[i];
					q[t][3] = q[h][3] + 1;
					f[x+fx[i]][y+fy[i]] = true;
					if(x+fx[i]==e1&&y+fy[i]==e2)
					{
						cout<<q[t][3] - 1;
						return 0;
					}
				}

				x = x + fx[i];
				y = y + fy[i];// 跳到该点，看下一个点
			}
		}
		h++;
	}
	return 0;
}
```

思路：

沿着最外层的一圈遍历一遍，如果有 '0' 就从该点开始深搜将所有经过的点都标记为 ''

被 '' 围住的 '0'，是一定搜索不到的

最后只要看还剩几个 '0'

```
#include<bits/stdc++.h>//4-1913  拯救指挥部  javacn
using namespace std;

char a[510][510];
int n,m;
// 方向的变化
int fx[5]= {0,0,1,0,-1};
int fy[5]= {0,1,0,-1,0};

// 深搜值为 '0' 的连通块，将其标记为 '*'
void dfs(int x,int y)
{
	a[x][y] = '*';
	int tx,ty;
	for(int i = 1;i <= 4;i++)
	{
		tx = x + fx[i];
		ty = y + fy[i];
		// 在迷宫内，且为 '0'（为了迷宫不可能是 '0'，值是 '\0' 也就是整数 0）
		if(a[tx][ty] == '0')
		{
			dfs(tx,ty);
		}
	}
}
```

```
int main()
{
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cin>>a[i][j];
		}
	}

	// 沿着外层的一圈，找到’0’从该点深搜
	for(int i = 1;i <= n;i++)
	{
		// 如果是第 1 行，或者最后一行，遍历列
		if(i == 1 || i == n)
		{
			for(int j = 1;j <= m;j++)
			{
				if(a[i][j] == '0')
				{
					dfs(i,j);
				}
			}
		}
		else
		{
			// 如果是该行的第 1 个元素、最后一个元素
			if(a[i][1] == '0') dfs(i,1);
			if(a[i][m] == '0') dfs(i,m);
		}
	}
```

```
	int c = 0;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			if(a[i][j] == '0')		c++;
		}
	}

	cout<<c;
	return 0;
}
```